Stakeholder Specific Visualisation from Heterogeneous Modeling Tools

Abdil Kaya[1], Stefan Dutre[2], Jef Stegen[2], Satya Prakash Jha[2], and Joachim Denil[1,3]

[1] University of Antwerp, Antwerp, Belgium
`Abdil.Kaya@student.uantwerpen.be, Joachim.Denil@uantwerpen.be`
[2] Siemens Industry Software, Leuven, Belgium
`firstname.lastname@Siemens.com`
[3] Flanders Make, Belgium

Abstract. Our software-intensive and cyber-physical systems are becoming more complex because of added features. Model-Based Systems Engineering helps to manage the complexity by introducing models and analyses in a top-down design of the system. Appropriate formalisms and tools are used at each abstraction level to model requirements, architecture and behaviour. Most of these tools offer different visualisations of the models. However, there is not a lot of support in the visualisation of the design that spans different modelling formalisms and tools. In this paper we propose a solution based on domain-specific languages. Our language defines the operational steps needed to combine (filtered) model elements together from different tools and map the resulting graph or tree to a visualisation template.

1 Introduction

Our software-intensive and cyber-physical systems are becoming more complex because of added features such as autonomous decision making, energy efficiency, and safety requirements. This results in a systems design process where the number of requirements (and thus later in the process, the number of components) is increasing, the heterogeneity of the components that need to be combined is bigger, and the number of stakeholders involved in the design is larger.

Model-Based Systems Engineering tries to help in managing the complexity by introducing models and analyses in a top-down design of the system. Architectural models are used during the decomposition of the system while simulation at different levels of abstraction/approximation allows for system-level evaluations. However, the heterogeneity in the design results in a broad set of formalisms being used, usually implemented in their own tool. For example, geometric models to allow for finite element analysis can be modelled in NX Nastran, requirements are modelled within tools such as Polarion, AMESim for 1D multi-physics modelling of the plant, etc. Each of these tools offers their own visualisation of the model and/or its behaviour.

CEUR-WS.org/Vol-2245/flexmde_paper_2.pdf

Nonetheless, different stakeholders would often benefit from creating customised visualisations that span multiple models and tools. For example, a safety engineer might benefit from a visualisation that shows a combination of safety requirements linked to changes in the architecture to immediately see which parts of the architecture are changed in recent developments and how this might impact the safety functions. A quality engineer might benefit from a visualisation that links requirements, architecture, and test results and metrics that allows him/her to see where problems in quality might occur. Mid level managers might want to see requirements or architecture models together with data from version control systems to evaluate which parts are worked on by whom.

However, the flexibility of creating visualisations over different models, modelling tools, and other process data sources is quite low. Visualisations are often offered within a single tool environment where the link with other artefacts within the design project is hard or even impossible. This initial work tries to provide a small domain-specific language that allows stakeholders to create their own personal visualisation based on a simple set of operations.

The rest of this paper is organised as follows: Section 2 introduces the running example used throughout the paper. Afterwards, Section 3 describes our preliminary approach to create stakeholder specific visualisations. Next, we discuss our approach and its current implementation in Section 4. Section 5 provides an overview of related work. Finally, we conclude in Section 6.

2 Running Example

New aircraft designs have shown a tendency towards More Electric Aircraft (MEA) for some decades now [4]. The MEA architecture is a phase between conventional aircraft architecture and the "all-electric" architecture. One of the two main focuses of the MEA is to replace the electro-hydraulic actuators by electromechanical actuators as primary and secondary control surface actuators.

In electro-mechanical actuators, an electric motor is directly responsible for the drive of an actuator. Because of safety regulations in aeronautics, it is important to develop jam-free EMA technologies in order to benefit from its advantages and role in reducing production and maintenance costs [10].

Our running example is to create a domain-specific visualisation for a safety engineer in the development an aeronautical actuator for load alleviation in aircraft wings. This visualisation allows the engineer to see the evolution of the requirements between different versions/time-stamps. This choice was guided by a the safety expert working on the electromechanical actuator to check whether to allow him to do change impact analysis on the defined hazards and its ramifications on the architecture and safety functions.

In our running example, the Polarion tool captures the requirements. Polarion is an Application Lifecycle Management (ALM) tool by Siemens with support for iterative development, continuous integration and automated testing. 'Work item' is an encompassing general term in Polarion for requirements, architectural components and test cases alike [1]. In this paper we will use 'work item' and 'requirement' interchangeably as a reference to requirements. A second source of information in our running example is a Subversion version control repository. The Subversion change logs can be used to detect changes between versions and/or dates.

At the time of writing, there are 983 requirements for the design of the actuator. An example of such requirement is as follows: *The Ball Screw of the Electro-Mechanical Actuators of Wing Ailerons should have a weight of less than x grams.*

3 Approach

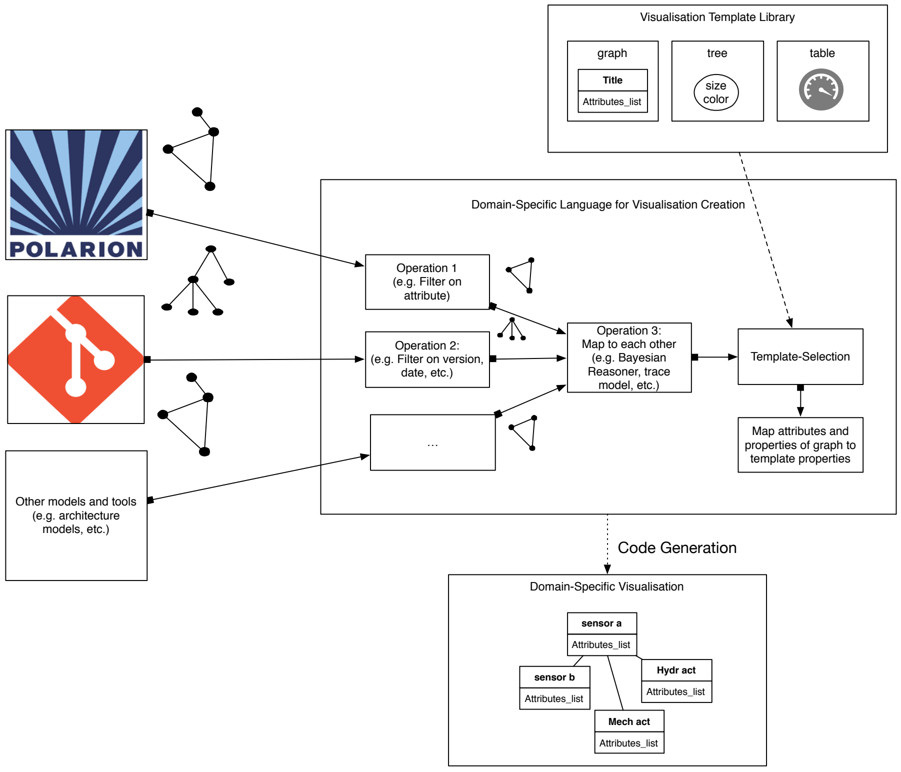

Visualisation Template Library
graph
Title
Attributes_list
tree
size
color
table
POLARION
Domain-Specific Language for Visualisation Creation
Operation 1 (e.g. Filter on attribute)
Operation 2: (e.g. Filter on version, date, etc.)
...
Operation 3: Map to each other (e.g. Bayesian Reasoner, trace model, etc.)
Template-Selection
Map attributes and properties of graph to template properties
Other models and tools (e.g. architecture models, etc.)
Code Generation
Domain-Specific Visualisation
sensor a
Attributes_list
sensor b
Attributes_list
Hydr act
Attributes_list
Mech act
Attributes_list

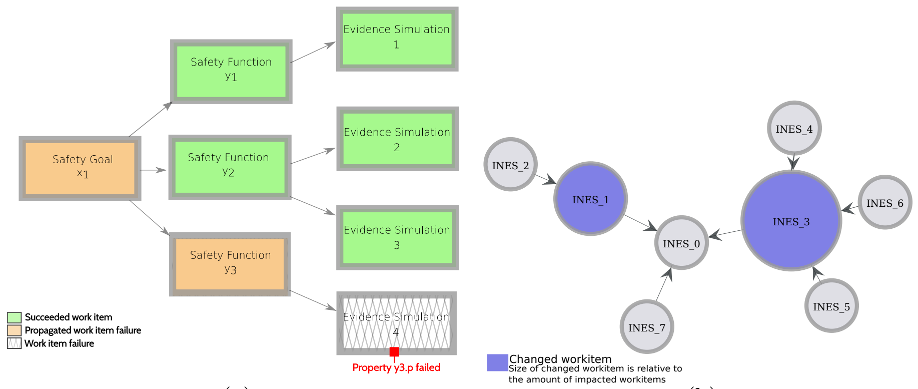
Fig. 1: Overview of the approach

The goal of this work is to develop a method to generate meaningful visualisations that span a multitude of formalisms and tools of the design of a complex system. Our approach uses a domain-specific language to empower stakeholders in visualising their specific concerns. Figure 1 shows an overview of of our domain-specific visualisation approach.

The approach starts by identifying various data sources. The data can range from design contracts, specifications, models, and repositories with a collection of various data. These sources are in short the providers of the model elements and all the data that can be traced to them. After this, a domain-specific language (DSL) provides support of different operations to be performed on the subset of work items. The selection of elements to keep or remove is modelled by the stakeholder. Next, the remaining model elements (or data) from the different sources has to be linked to each other. Afterwards, the linked data (graph, forest, tree, etc.) have to mapped to the visualisation template and its attributes. Finally, the stakeholder-specific visualisation is created, which is representative of his concerns. In our case, the visualisation in python code.

3.1 Templates

The visualisation library templates implement a number of general approaches for visualising the combined model elements. They are flexible in the sense that the stakeholder can either define one themselves, or build up on existing templates. Figure 2 shows different visualisations using different visualisation templates. Figure 2a shows an impact analysis of a changed safety requirement. It includes the automatic simulation and testing of the safety properties to evaluate if the system is still safe. The template is a tree-layout where the properties that can change are the labels of the boxes and the color of the boxes. Figure 2b shows another type of impact analysis where the size of the impact is reflected in the size of the circle. The template is a graph template with different shapes. The color, size and label are attributes of the visualisation elements.



(a) (b)

Fig. 2: Visualisation template examples

3.2 Domain-Specific Language

Using a domain-specific language allows for problems to be solved closer to the problem domain, instead of the solution domain. As such, a stakeholder with their particular domain knowledge can generate a meaningful visualisation, without having to understand the underlying workings of visualisations, which is the solution domain. This means that the gap between a user's mental model requires fewer steps to get to the result. It will form a more intuitive approach to requesting the visualisations one is interested in. Rather than having a transformation expert study the domain space, it is easier for the domain expert to become an expert given the right tools. DSLs try to close this cognetive gap between the problem and the solution [11, 8].

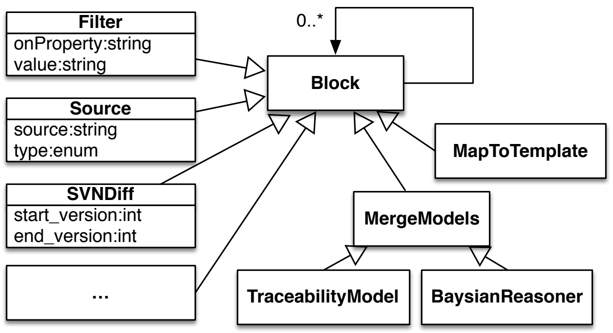


Fig. 3: Language definition of the visualisation generation language

Figure 3 shows our initial language definition for the visualisation tool. The concept is based on causal-block diagrams [6]. Causal-block diagrams use blocks and connections between blocks to depict the data-flow between different mathematical operations. In our case, we use the blocks to define operations on the source data and to mould it towards the stakeholders visualisation requirements.

The *Source* block allows to acquire the data that we need to create the heterogeneous model views. In our current approach we created custom parsers for the different modelling tools involved in the visualisation approach. Polarion is our primary source for requirements. It integrates a PostgreSQL database for the server its back-end data storage and provides different interfaces for other tools to interact with. The Polarion approach to requirements management has support for text-based, tool-centric as well as a hybrid definition of requirements. The change logs of Subversion are parsed with regular expressions in order to detect which work items are adapted and what their location is within the repository.

MergeModel blocks allows us to combine the different heterogeneous data-sources. An explicit traceability model allows for the linking of the same elements together. In our case the name of the requirement is explicitly traced to a file with the same name (and hierarchy) in the SVN. If no such trace model is available, reasoners such as a Baysian reasoner could be used to link the data together [7].

Different operation blocks are available to modify the context of the model. For example, the *Filter* block allows the modeller to filter out specific blocks in the model based on an attribute or expression. In our running example, the filter is applied to filter out only the items that are tagged with "Author" tag in the Polarion requirements tools. The *SVNDiff* block looks at the SVN-data that has been linked with a requirement and compares different versions. It tags the elements with a property "new, modified, unchanged or deleted" that can be visualised later.

Finally, the *MapToTemplate* allows the engineer to specify how the element types and properties map to visualisation types (e.g. shapes) and visualisation properties (e.g. size, color, text, etc.). In our visualisation, the forest of filtered requirements are mapped to a tree visualisation. The label of the box is mapped to the requirement ID. The color of the box is mapped to the SVNDiff properties: green represents a new requirement, orange represents a modified requirement, red marks that the requirement is deleted and grey shows that the requirement is unchanged between two different versions.

3.3 Code generation

We have chosen to generate code to visualise the different model elements and its template. Because of its rapid deployment and eased maintenance, we have chosen to use Python. This interactive visualisation tool is developed and implemented using a modified Cairo [4] library and the GTK+ library. Listing 1.1 shows the generated code for filtering Polarion requirements based on the specified properties in the model. In the example, only workitems that are of four specific types are kept in the model.

Listing 1.1: Requirements Filter

```
def filter_main_reqs(workitems_index):

for v in g[workitems_index].vertices():
  if "sslrequirement" not in \
  workitems[workitems_index][int(v)].type and \
  "colrequirement" not in \
  workitems[workitems_index][int(v)].type and \
  "ailrequirement" not in \
  workitems[workitems_index][int(v)].type and \
  "syslrequirement" not in \
  workitems[workitems_index][int(v)].type:
    hidden[workitems_index][v] = True
g[workitems_index].set_vertex_filter(hidden[workitems_index], inverted=True)
```

[4] https://www.cairographics.org

3.4 Results

In our running example, we used the needs of a safety engineer as reference. The model for generating the visualisation is shown in Figure 4The visualisations and functionalities presented below reflect this decision. Our running example aimed at visualising the difference between different versions of a subset of the requirements. Figure 5 shows the generated visualisation of the running example. The template is a tree-template, as it matches the hierarchy of the requirements. The colors shows the modified, deleted and added requirements in a single visualisation. This removes the burden on the safety engineer to go through the change logs or requirements repository.

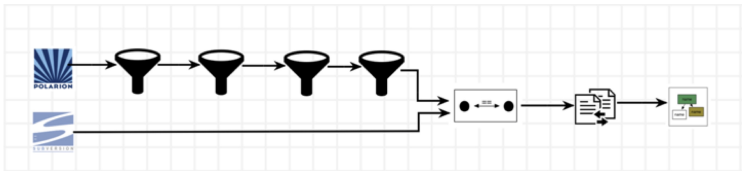
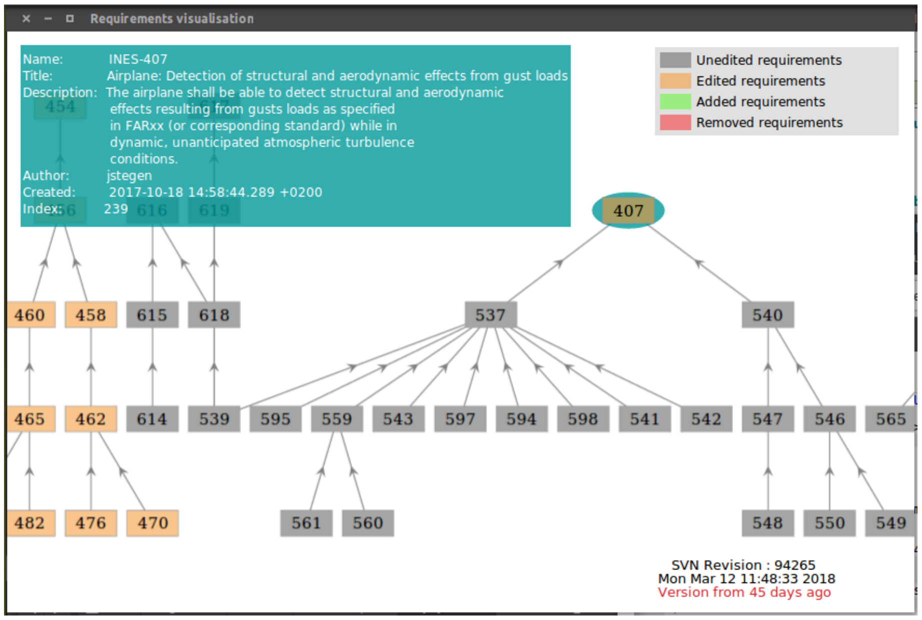


Fig. 4: Domain-specific model of the visualisation



Name: INES-407
Title: Airplane: Detection of structural and aerodynamic effects from gust loads
Description: The airplane shall be able to detect structural and aerodynamic effects resulting from gusts loads as specified in FARxx (or corresponding standard) while in dynamic, unanticipated atmospheric turbulence conditions.
Author: jstegen
Created: 2017-10-18 14:58:44.289 +0200
Index: 239

⬜	Unedited requirements
🟧	Edited requirements
🟩	Added requirements
🟥	Removed requirements

SVN Revision : 94265
Mon Mar 12 11:48:33 2018
Version from 45 days ago

Fig. 5: Interactive, event-driven requirements visualisation

4 Discussion

When acquiring data, more attempts should be made towards using standardized data exchange formats. In our case, we defined custom parsers for acquiring the SVN data and the Polarion data. Defining custom parsers for each available tool is not viable. Therefore, standard exchange methods should be used to get the data from the different tools. Open Services for Lifecycle Collaboration (OSLC) is such a data linking technology [14]. OSLC defines a set of specifications that enables collaboration of different heterogeneous tools. OSLC builds upon the W3C Resource Description Framework (RDS) and RESTful webservices. Polarion natively supports OSLC as part of its interface. However, a lot of commercial tools do not support OSLC. In that case, wrappers can be used to expose the information in the models, as shown for the Simulink tool in [3].

The first version of our domain-specific language is very adapted towards solving certain visualisation problems within the scope of the safety engineer. However, the language should be adapted so it contains a lot more operations that are useful to the engineers defining a visualisation. For example, even complex operations such as running the integrations tests or running a simulation and getting its results are useful to visualise. More templates should also be added, e.g. a template with gauges can show the progress of certain work-parts or the passing of tests. We will extend the language with these operations as new case studies for visualisation are done.

The usability of the language is not very high at this point. The engineer has to pay attention that the properties to filter on are correctly spelled. Our syntax directed editing tool, created with AToMPM [12], allows to fill up different filter lists that the user can select, while parsing the models. This allows for more interactivity and easier to create models. Furthermore, we should define more static constraints such that types between the blocks are correctly matched (e.g. graph, tree, etc.) and that certain blocks can only work on certain input data. It should also notify the user why a certain visualisation template is not advisable (graph structure in model versus tree visualisation). Ports can help in defining these constraints easier. In the future we will focus on adding such constraints and type checks.

5 Related work

The work closest to our approach is presented by Basole et al. [2]. They propose a visualisation tool which focuses on early design phases in CES (Complex Engineered Systems). There are several strong parallels in our visualisation DSL and the tool created in Basole et al. Both tools want to resolve industry needs in the growing complexity and interdependency of artefacts in MBSE design methods. Whereas the tool proposed in Basole et al. offers the stakeholders a graphical user interface to filter and define their visualisation, it is a closed tool in which the stakeholder can only select the attributes that are present in the tool. We want to create an open language to create visualisations that easily extendible.

Within a single tool there are DSLs to create a stakeholder specific visualisation. Kitalpha is such a domain-specific language for the Capella workbench, specialised in the description of architecture and with support up to architecture evaluation. Albeit on a different phase in development, the focus in terms of separation of concerns is similar to ours. Its approach is twofold, one DSL is used for the description of the architecture framework and another for the description of viewpoints. Langlois et al. conclude that using a DSL and code generation for the description of the architecture framework as well as the viewpoints have resulted in efficiency and autonomy for the designers and developers [9]. The difference in comparison with the vision of our work is that our tool assumes an external approach to systems engineering tools. This leads to and requires a more generic approach. Feather et al. [5] acknowledges that decision-making in the early phases of system its development remains a human-driven task, because there is no alternative to domain expertise. The authors have experimented with TreeMap visualisations for requirements. These TreeMaps can be used to visualise hierarchy with position, relative importance with surface area, and the attainment with color.

Lastly, Weber and Weisbrod note the importance of user-adaptable views to guide the specification process [13]. Requirements engineering is a highly dynamic and crucial engineering activity. They stress that the provision of such views by workflow management tools cannot simply be the product of attribute filters, but should also support dynamic changes and automatic display of derived information.

6 Conclusion

Stakeholder-specific visualisations over the bounds of tools are useful for engineers to immediately observe important aspects of the design. However, currently there are not a lot of methods or tools available to allow such visualisations. In this work we tried to mend this gap by providing engineers with a domain-specific language to visualise their interests in the design. While, it is still a preliminary language and implementation, we have shown that complex visualisations can be created. Our approach uses a data-flow language to shape the different models and merge them. Finally, the resulting graph or tree is mapped to a visualisation template such that model elements and its properties are mapped to visualisation elements and its properties.

References

1. Help - Polarion ALM Platform. https://almdemo.polarion.com/polarion/help/?topic=/com.polarion.xray.doc.user/ugchReqEngineers.html, 2018.
2. Rahul C. Basole, Ahsan Qamar, Hyunwoo Park, Christiaan J.J. Paredis, and Leon F. McGinnis. Visual Analytics for Early-Phase Complex Engineered System Design Support. *IEEE Computer Graphics and Applications*, 35(2):41–51, March 2015.

3. Matthias Biehl, Jad El-Khoury, Frédéric Loiret, and Martin Törngren. On the modeling and generation of service-oriented tool chains. *Software & Systems Modeling*, 13(2):461–480, 2014.
4. Stefan Dutré. Innovation in the Development of Electronic Systems For Aeronautics, January 2017.
5. M. S. Feather, S. L. Cornford, J. D. Kiper, and T. Menzies. Experiences using Visualization Techniques to Present Requirements, Risks to Them, and Options for Risk Mitigation. In *2006 First International Workshop on Requirements Engineering Visualization (REV'06 - RE'06 Workshop)*, pages 10–10, September 2006.
6. Cláudio Gomes, Joachim Denil, and Hans Vangheluwe. Causal-block diagrams. Technical report, University of Antwerp, 2016.
7. Sebastian JI Herzig and Christiaan JJ Paredis. Bayesian reasoning over models. In *11th Workshop on Model Driven Engineering, Verification and Validation*. CEUR, 2014.
8. Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, April 2008.
9. Benoit Langlois, Daniel Exertier, and Boubekeur Zendagui. Development of Modelling Frameworks and Viewpoints with Kitalpha. pages 19–22. ACM Press, 2014.
10. J.A. Rosero, J.A. Ortega, E. Aldabas, and L. Romeral. Moving towards a more electric aircraft. *IEEE Aerospace and Electronic Systems Magazine*, 22(3):3–9, March 2007.
11. Laurent Safa. The Practice of Deploying DSM - Report from a Japanese Appliance Maker Trenches. In *Proceedings of the 6th OOPSLA Workshop on Domain Specific Modeling*, 2006.
12. Eugene Syriani, Hans Vangheluwe, Raphael Mannadiar, Conner Hansen, Simon Van Mierlo, and Huseyin Ergin. Atompm: A web-based modeling environment. In *Joint proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition co-located with the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013): September 29-October 4, 2013, Miami, USA*, pages 21–25, 2013.
13. Matthias Weber and Joachim Weisbrod. Requirements engineering in automotive development-experiences and challenges. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pages 331–340. IEEE, 2002.
14. OSLC Core Specification Workgroup. Oslc core specification version 2.0. *Open Services for Lifecycle Collaboration, Tech. Rep*, 2010.

Acknowledgments

This work is partially supported by the European Eureka project : Innovations in the development of electronic systems for Aeronautics (INES) and the Flemish VLAIO project INES.